**Khushveen Kaur Umra**

**28th June 2022**

# SE 317: Lab 6

## Instructions

## Boundary Conditions: The Correct Way

Book Pages (for additional description): Chapter 7: Page 79, 80, 81
**Use the source code provided in the zip folder**. There are six classes in the zip folder.

1. Bearing.java
2. BearingOutofRangeException.java
3. BearingTest.java
4. Rectangle.java
5. Rectangletest.java
6. ConstrainsSideTo.java

Lab objective: To understand the concepts of throws declaration and try/catch method.
Some classes have errors. You need to find and fix the errors, then submit the screenshots of the corrected code by using two different methods: "throws" method and "try/catch method". This assignment will also help understand how you can do unit testing at different boundaries and how "range" works.

**Steps:**

1- Run the BearingTest.Jav code
2- You will get error messages as in figure 1 below
3- Inspect the BearingTest.java, it has 3 functions:
   i. `public void answersValidBearing()`
   ii. `public void answersAngleBetweenItAndAnotherBearing()`
   iii. `public void angleBetweenIsNegativeWhenThisBearingSmaller()`

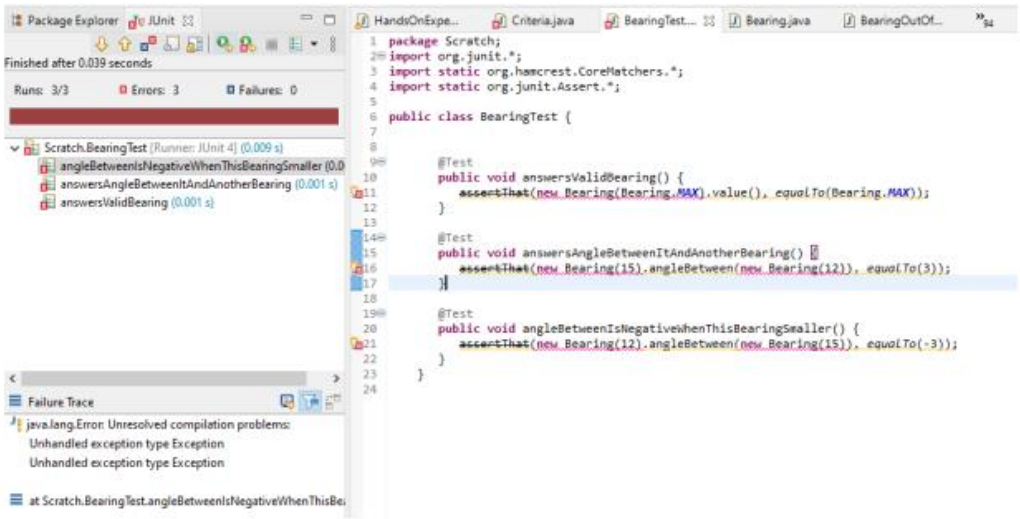



Fig 1

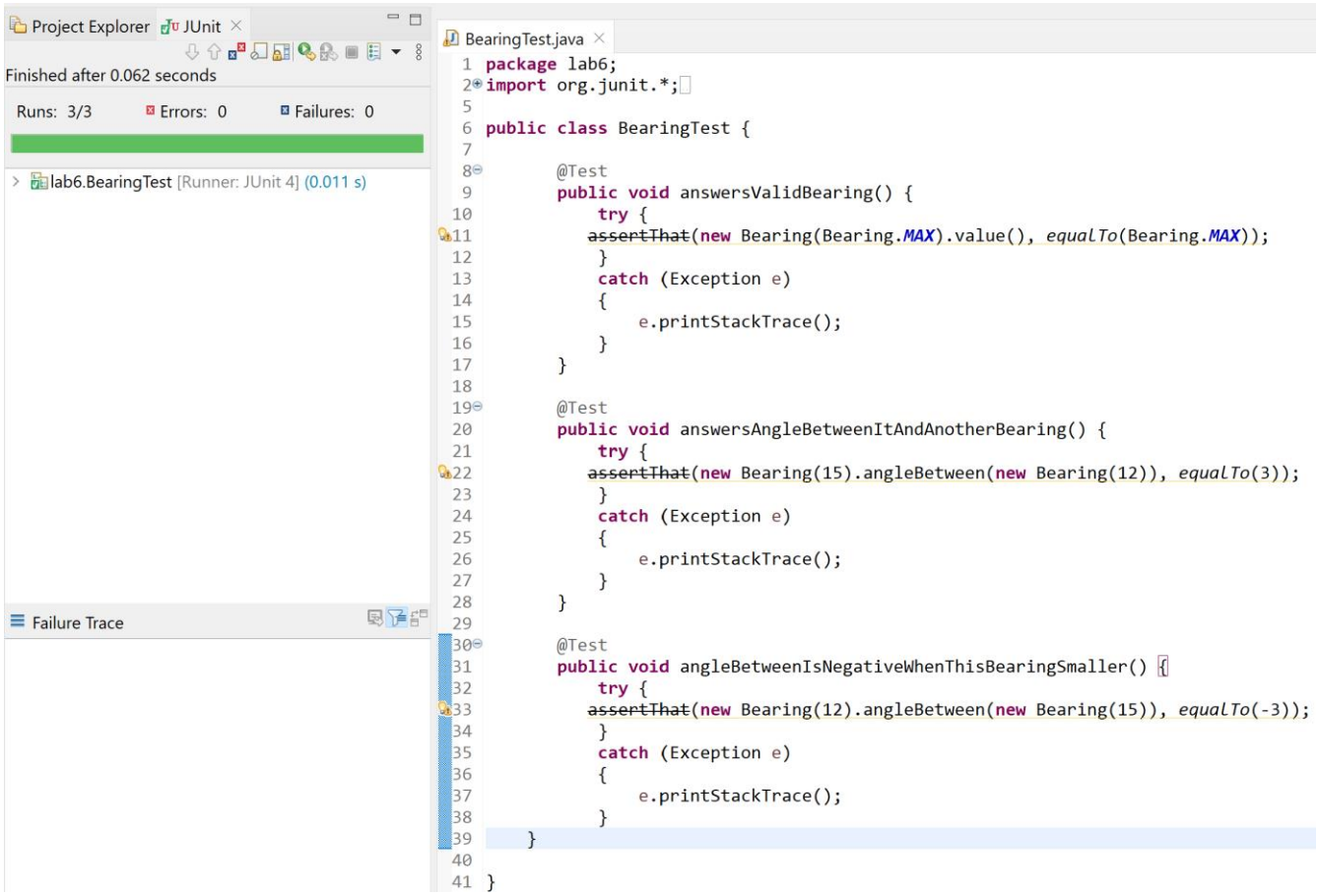
1. **TODO:**

## Part 1

These 3 functions contain some errors. You need to fix the by using both **throws** method and **try/catch** method.

1- First, use **throws** method to fix the code. When you finish, take the screenshot of the passed result with your code


2- Next, Replace the throw exception with the "**try/catch**" method, run the code again, and submit the screenshot of the **passed** result with your code




In both cases, when you take the screenshots, make sure you also take screenshot of the BearingTest.java code so that we can see your code.

## Part 2

1- After fixing the code, inspect the answersAngleBetweenItAndAnotherBearing() function and the `angleBetweenIsNegativeWhenThisBearingSmaller().`

Analyze the code in Bearing.java.

Note: A circle has 360 degrees in either direction (clockwise or counter clockwise). Rather than storing the direction of a travel as a native type, Bearing.java encapsulates the direction along with logic to constrain its range.

### 2. TODO:

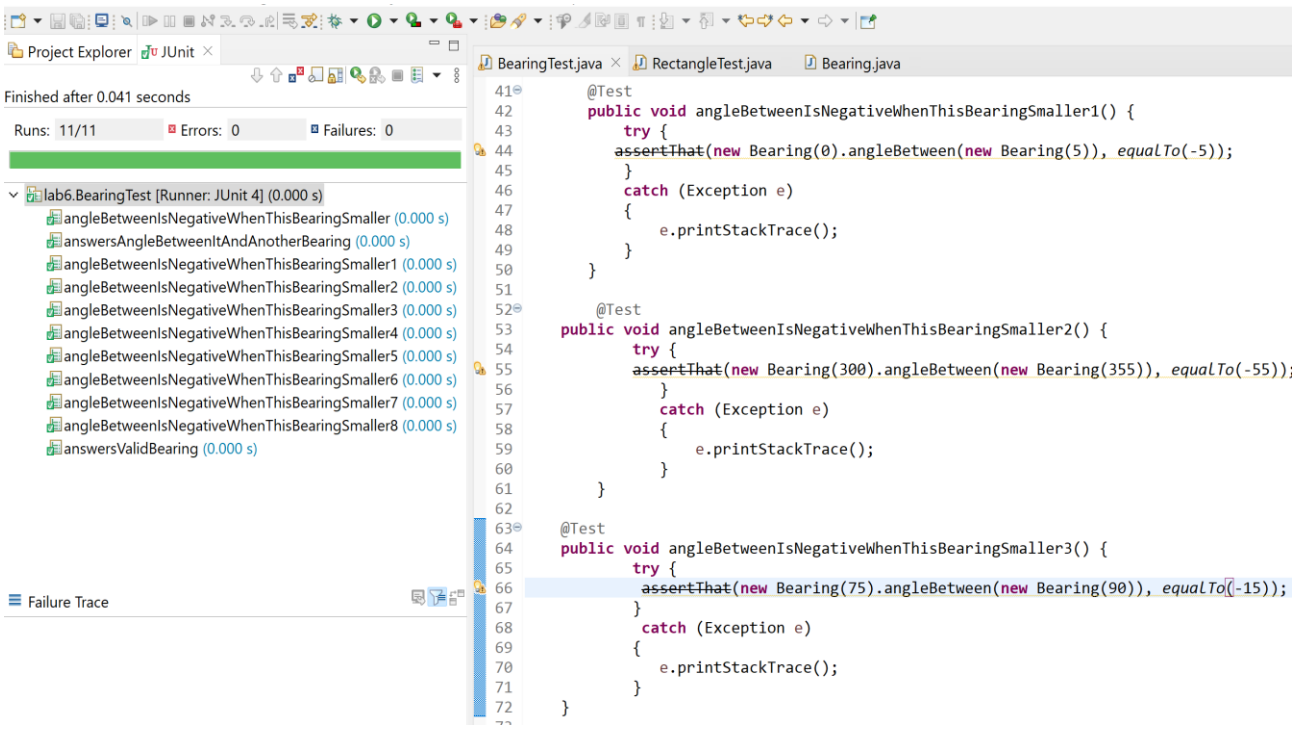
**Write 8 test cases** similar to angleBetweenIsNegativeWhenThisBearingSmaller() functions and use try/catch method or throws function (either one) and make sure the test cases pass. Take a screenshot of the test cases. Your test cases should test different bearings (0, 355, 90, 55, 100, 12, 123, etc.)

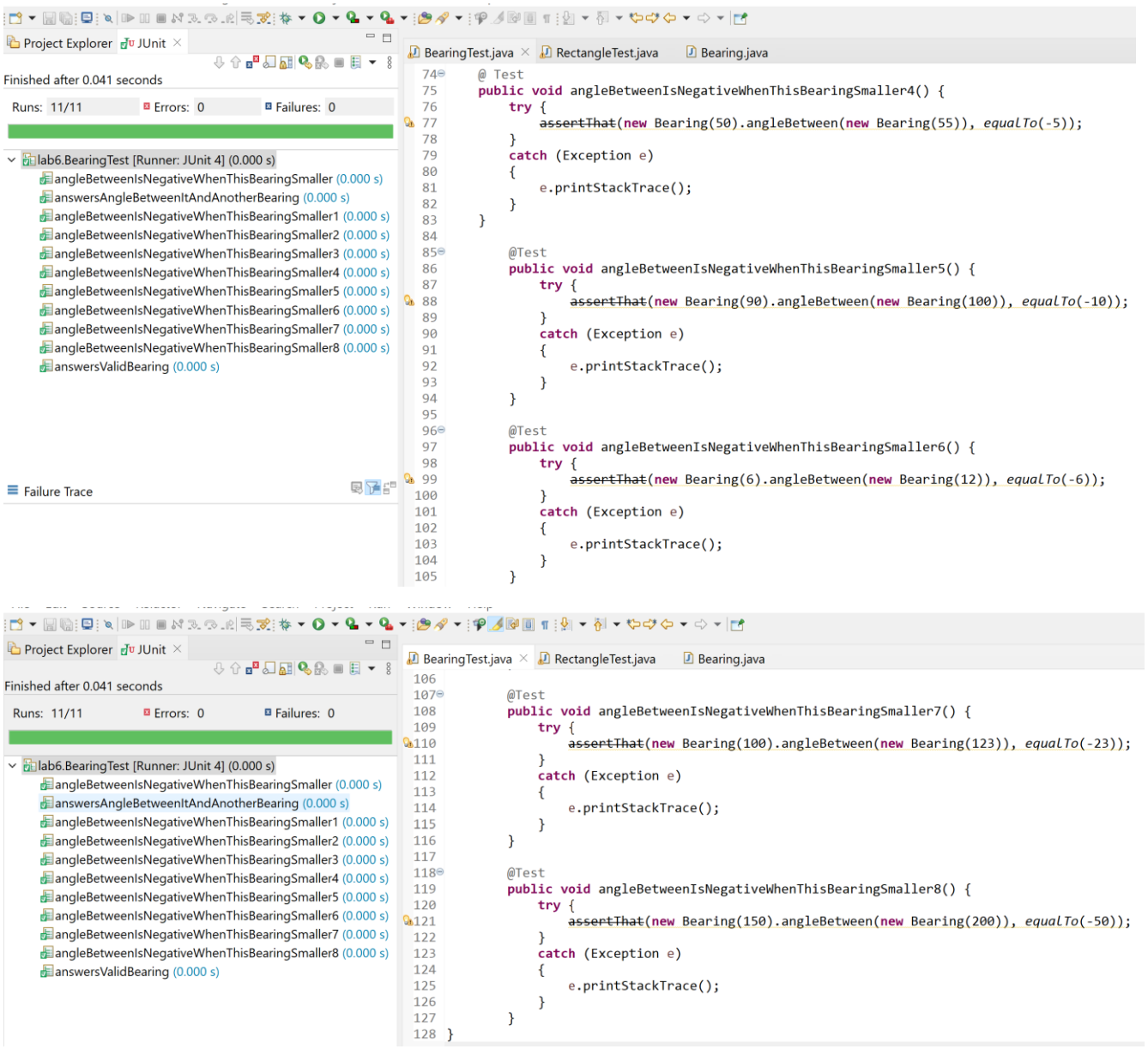
Hint: Inspect bearing.java code to see how it works. Create similar test cases and take a screenshot of test cases and make sure it passes.

Example:-
Start with similar test case of angleBetweenIsNegativeWhenThisBearingSmaller() function.
**Note,** this example uses Throws Method but you can use any method. See below.

```
@Test
public void angleBetweenIsNegativeWhenThisBearingSmaller2() throws Exception
{
	assertThat(new Bearing(5).angleBetween(new Bearing(15)), equalTo(-10));
}
```

Note that angleBetween() returns an int. We are not placing any range restrictions on the result.









## Part 3

Inspect the classes Rectangle and RectangleTest from Lab 6 zip folder

Some constraints might not be as straightforward. Suppose we have a class that maintains two points, each point is an (x, y) integer tuple. The **constraint** on the range is that the two points must describe a **rectangle** with no side greater than 100 units. That is, the allowed range of values for both x, y pairs is interdependent.

We want a range assertion for any behavior that can affect a coordinate, to ensure that the resulting range of the x, y pairs remains legitimate—that the *invariant* on the Rectangle holds true.

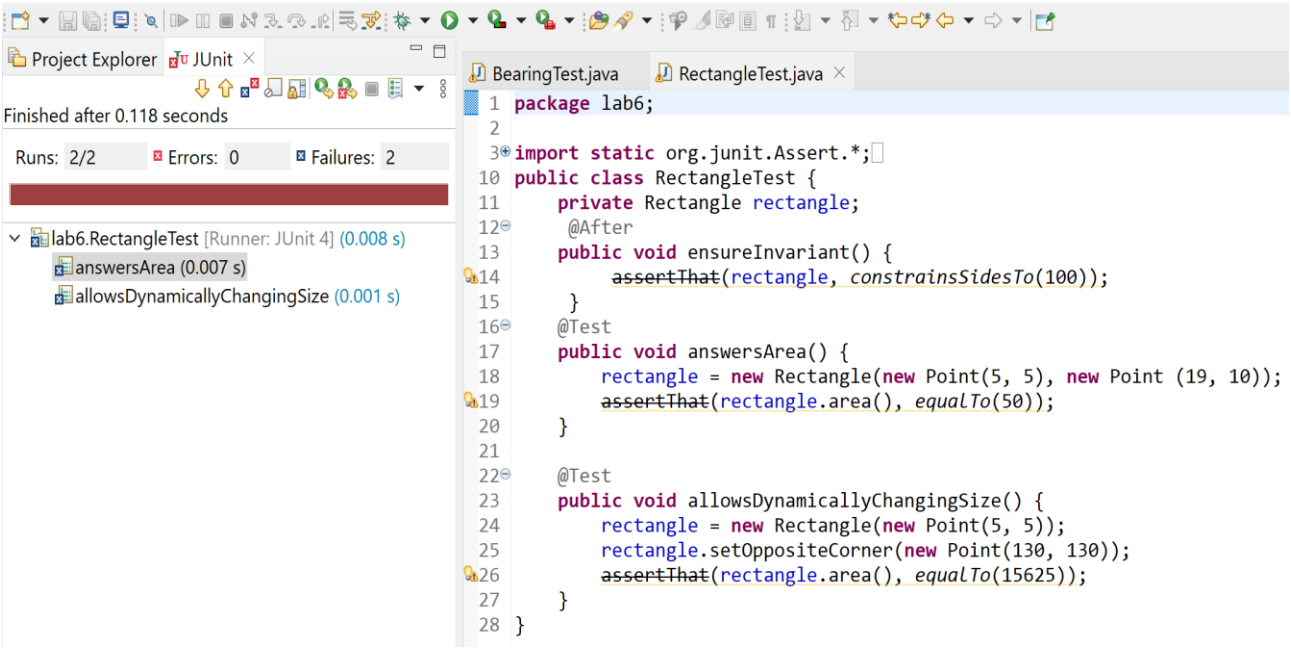
More formally: an **invariant** is a condition that holds true throughout the execution of some chunk of code. In this case, we want the invariant to hold true for the lifetime of the Rectangle object—that is, any time its state changes.

We can add invariants, in the form of assertions, to the @After method so that they run upon completion of any test. An implementation for the invariant for our constrained Rectangle class looks like RectangleTest in the source code folder.

3. **TODO:**

Run the test cases in RectangleTest.Java.

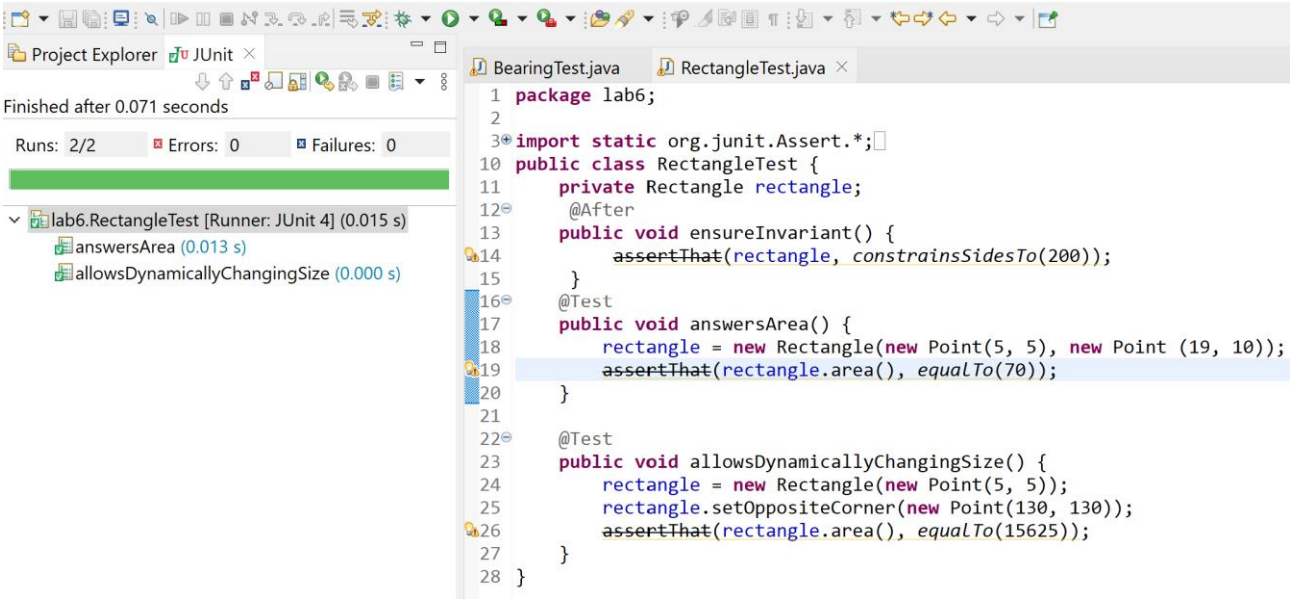
Any error? Take a screenshot of your code output

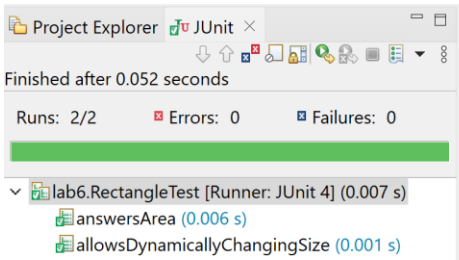



4. **TODO:** Fix the error(s) of the code and run the code again.

Take a screenshot of your code and output

**Hint:** Inspect Rectangle.java and look at the function public int area(). Analyze it.






BearingTest.java RectangleTest.java

```
1  package lab6;
2
3  import static org.junit.Assert.*;
10 public class RectangleTest {
11     private Rectangle rectangle;
12      @After
13     public void ensureInvariant() {
14          assertThat(rectangle, constrainsSidesTo(100));
15      }
16     @Test
17     public void answersArea() {
18         rectangle = new Rectangle(new Point(5, 5), new Point (19, 10));
19         assertThat(rectangle.area(), equalTo(70));
20     }
21
22     @Test
23     public void allowsDynamicallyChangingSize() {
24         rectangle = new Rectangle(new Point(5, 5));
25         rectangle.setOppositeCorner(new Point(105, 105));
26         assertThat(rectangle.area(), equalTo(10000));
27     }
28 }
```

5. **TODO:**

Answer the following questions:

1. What is throw exception and how does it fix the code?

➔ Throw exception is a mechanism to throw the exception to the calling method to a level where it can be handled. It tells the user or the code that the provided input is not valid, or the actions being performed on any data or objects are not valid. By using the throw exception, the execution of the current function will stop, and the control will be passed to the first catch block in the call stack.

2. What is try-catch method and how does it fix the code?

➔ The 'try' statement allows the user to define a block of code that may throw an exception to be tested for errors while the code is being executed and the 'catch' statement allows the user to define a block of code to be executed if an error occurs in the corresponding 'try' block and catches the exception whenever a specific type of exception occurs. It fixes the code by not killing the script when an error occurs, but instead gives the user a chance to handle it in 'catch'.

3. Is there any difference between throw exception and try-catch method? If yes, explain.

➔ The only difference between them is that using the try-catch method allows the user to handle exception surrounding code that might raise an exception. Whereas, using 'throws' keyword, allows the user to simply declare exception that might raise from that specific method during execution.

➔ The 'throws exception' is used to indicate that particular exception is possibly thrown from executing method at run-time, whereas try-catch method is used to handle exception scenario.